iothnamed

a DNS server/forwarder/cache for the Internet of Threads

Renzo Davoli
University of Bologna (Italy)
VirtualSquare

DNS DevRoom

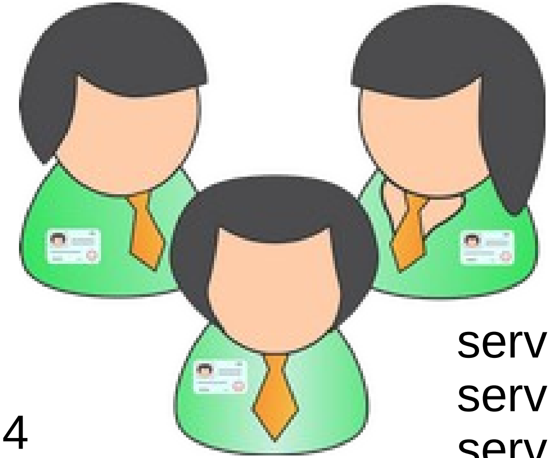
Internet of Threads (IoTh)

What is an end-node of the Internet?

It depends on what is identified by an IP address.

- Legacy approach:
  - Internet of Hosts – Internet of Network Controllers.
  - Internet of Virtual Machines – Internet of virtual Network Controllers
  - Internet of Namespaces
- Internet of Threads – IoTh – Processes/threads are autonomous *nodes* of the Internet

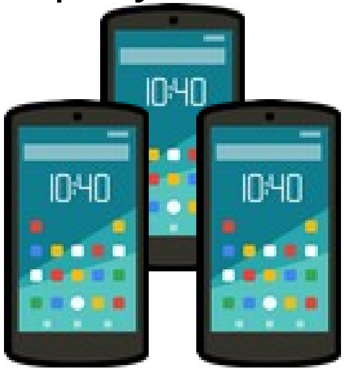
Internet of Threads (IoTh)



processes

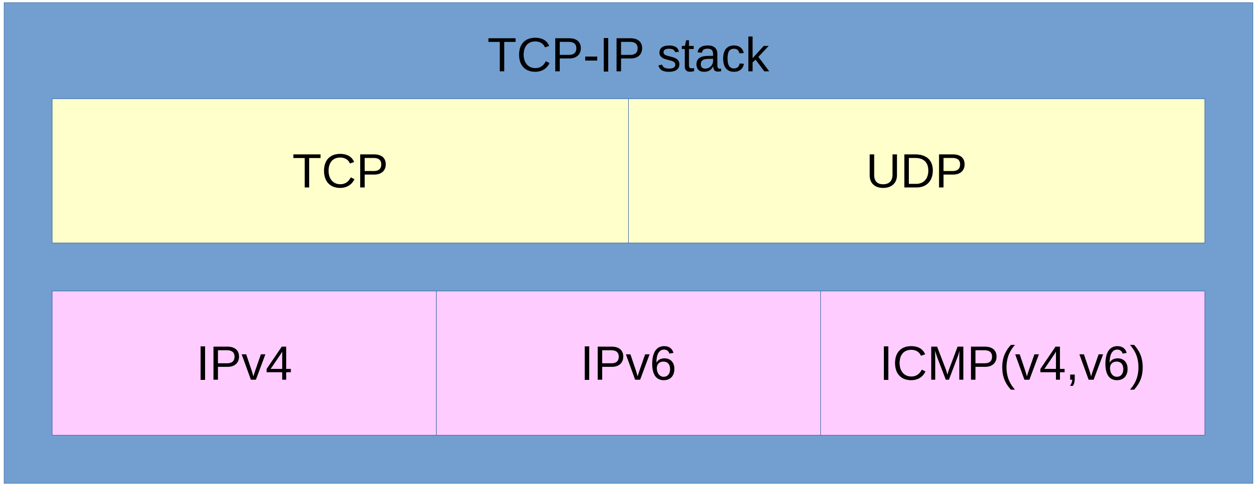
host.company.com→11.12.13.14

service1.company.com → 2001:1:2::1
service2.company.com → 2001:1:2::2
service3.company.com → 2001:1:2::3



Network Stack

- API to application layer



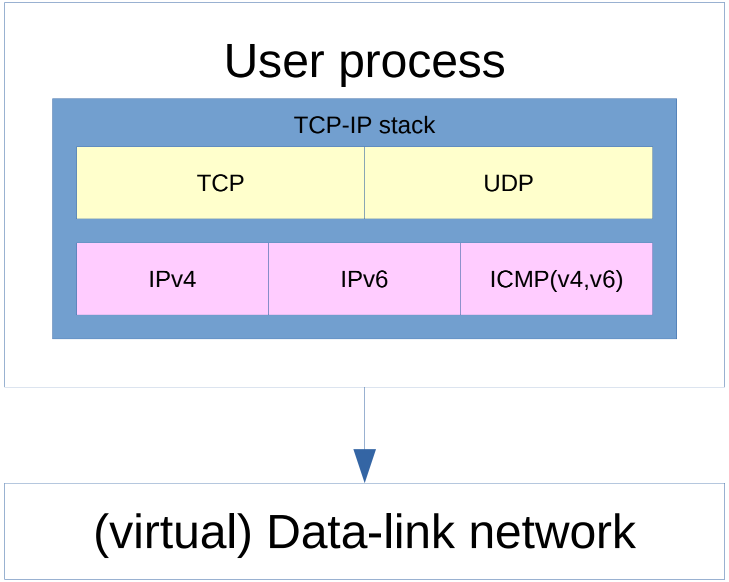
TCP-IP stack

TCP
UDP

IPv4
IPv6
ICMP(v4,v6)

- API (NPI) to data-link

IoTh


User process
TCP-IP stack
TCP
UDP
IPv4
IPv6
ICMP(v4,v6)
(virtual) Data-link network

libioth

- IoTh requirements:
  - TCP-IP stacks as libraries
  - Virtual Ethernet
- Libioth: one further step
  - A framework library
  - Actual TCP-IP stack implementations are loaded as plugins
  - Unified API
  - Currently supported stacks: kernel, vdestack, picox(picotcp), (WIP: lwip)
  - Libioth supports VDE as Virtual Ethernet.

API to the App Layer

- Complete
  - All currently available ops must be supported
- Usable
  - Syntax must be consistent with the major standards
- Minimal/Clean
  - Avoid useless or duplicated ops

API requirements

- Configuration calls
  - Create/Delete a stack instance
  - Configure parameters (as IP address-es, routing etc)
- Communication calls
  - open/close a communication endpoint
  - send/receive/set-get options

API design choices:
1- Stack creation/deletion

- A stack is identified by a descriptor of type:
  - `struct ioth *`
- Stack Creation:

```
struct ioth *ioth_newstack(const char *stack, const char *vnl);
struct ioth *ioth_newstackl(const char *stack, const char *vnl, ... /* (char  *) NULL */);
struct ioth *ioth_newstackv(const char *stack, const char *vnlv[]);
```

  - `ioth_newstack` for one interface (or none if `vnl==NULL`), the others are for more interfaces.
  - The string `stack` selects the stack implementation, loaded as a plugin.
  - `vnl` stands for “Virtual Network Locator”, it selects the VDE network to connect the virtual interface(s).
- Stack deletion:

```
int ioth_delstack(struct ioth *iothstack);
```

API design choices:
2- Communication

- Creation of a communication endpoint:

```
int ioth_msocket(struct ioth *stack, int domain, int type, int protocol);
```

  - It extends `socket(2)`, it allows the choice of the stack
  - It returns a *real* file descriptor that can be used in `poll(2), select(2)`. It is possible to wait for I/O events coming from devices, sockets, ioth_sockets using different net implementations….

- for everything else… Berkeley Sockets

```
ioth_close, ioth_bind, ioth_connect, ioth_listen, ioth_accept, ioth_getsockname, ioth_getpeername,
ioth_setsockopt, ioth_getsockopt, ioth_shutdown, ioth_ioctl, ioth_fcntl, ioth_read, ioth_readv, ioth_recv,
ioth_recvfrom, ioth_recvmsg, ioth_write, ioth_writev, ioth_send ioth_sendto and ioth_sendmsg
```

  - have the same signature and functionalities of their counterpart without the `ioth_` prefix

vpoll

- libioth’s file descriptors can be used in poll, select, ppoll, pselect, epoll…
- A way to generate arbitrary poll events was missing
- libvpoll uses a kernel module to provide a complete support (or implements an emulation able to manage POLLIN, POLLOUT and partially POLLHUP)

API design choices: 3- Configuration

- No more calls needed!
- Net configuration via AF_NETLINK sockets
- As defined in:
  - RFC 3549 - Linux Netlink as an IP Services Protocol
- *Helper libs are provided as services using the API (e.g. nlinline+ or iothconf).*

Example: legacy send “ciao\n”

```c
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char *argv[]) {
  struct sockaddr_in dst = {
    .sin_family = AF_INET,
    .sin_port = htons(5000),
    .sin_addr.s_addr = inet_addr("10.0.0.2")
  };

  int fd = socket(AF_INET, SOCK_DGRAM, 0);
  sendto(fd, "ciao\n", 5, 0, (void *) &dst, sizeof(dst));
  close(fd);
}
```

Example: ioth send “ciao\n”

```c
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ioth.h>

int main(int argc, char *argv[]) {
  struct ioth *stack = ioth_newstack("vdestack", "vde:///tmp/hub");
  struct in_addr myaddr = {.s_addr = inet_addr("10.0.0.1")};
  int ifindex = ioth_if_nametoindex(stack, "vde0");
  ioth_ipaddr_add(stack, AF_INET, &myaddr, 24, ifindex);
  ioth_linksetupdown(stack, ifindex, 1);

  struct sockaddr_in dst = {
    .sin_family = AF_INET,
    .sin_port = htons(5000),
    .sin_addr.s_addr = inet_addr("10.0.0.2")
  };
  int fd = ioth_msocket(stack, AF_INET, SOCK_DGRAM, 0);
  ioth_sendto(fd, "ciao\n", 5, 0, (void *) &dst, sizeof(dst));
  ioth_close(fd);
}
```

IoTh ecosystem Summary

- 1: nlinline
- 2: libnlq
- 3: iothconf
- 4: iothradvd
- 5: iothdns
- **6: iothnamed**
- 7: namedhcp
- 8 otip-utils

- **6: iothnamed**
  - Scenario 1: local+proxy
  - Scenario 2: delegated-subdomain
  - Scenario 3: localhash+proxy
  - Scenario 4: delegated+hash
  - Scenario 5: OTIP+proxy

IoTh ecosystem 1: nlinline

- Network stacks are generally considered as services provided by the kernel. So there are commands to configure the stacks like iproute or the old ifconfig.
- A library providing functions to configure a stack was missing
- nlinline is a light library of inline functions providing access to all the basic configuration ops (add/del an interface, set an interface up/down, add/del IP addresses, add/del routes). It uses netlink as described in RFC3549.
- Syntax “similar” to “ip” (iproute) commands or libc functions
- e.g.

```c
int ifindex = ioth_if_nametoindex(stack, "vde0");
ioth_ipaddr_add(stack, AF_INET, &myaddr, 24, ifindex)
```

IoTh ecosystem 2: libnlq

- Network stacks are generally considered as services provided by the kernel. A library able to decode configuration requests via netlink was missing.
- libnlq forges and decodes rt-netlink messages both at client and at server side.
- It also provides an emulation layer to support deprecated (though still used by glibc) netdevice ioctl.

IoTh ecosystem 3: iothconf

- Internet of Threads (IoTh) stack configuration made easy peasy
- iothconf can use four sources of data to configure an ioth stack
  - static data (IPv4 and/or IPv6)
  - DHCP (IPv4, RFC 2131 and 6843)
  - router discovery (IPv6, RFC 4861)
  - DHCPv6 (IPv6, RFC 8415 and 4704
- Es:

```
struct ioth *stack =
    ioth_newstackc("stack=vdestack,vnl=vxvde://234.0.0.1,eth,ip=10.0.0.1/24,gw=10.0.0.254");

struct ioth *stack = ioth_newstackc("stack=vdestack,vnl=vxvde://234.0.0.1,eth,dhcp");

struct ioth *stack =
    ioth_newstackc("stack=vdestack,vnl=vxvde://234.0.0.1,auto,fqdn=host.v2.cs.unibo.it");
```

IoTh ecosystem 4: iothradvd

A Router Advertisement Daemon for the Internet of Threads

- iothradvd is a router advertisement daemon for IPv6. It listens for router solicitation messages and sends router advertisements as described in "Neighbor Discovery for IP Version 6 (IPv6)" (RFC 4861). Hosts can automatically configure their addresses, prefixes and other parameters using the values acquired by RA messages.
- iothradvd is a daemon in the Internet of Threads definition: given that the process is a network node by its own, iothradvd runs as a thread.

IoTh ecosystem 5: iothdns

- The domain name resolution functions provided by the C library use the TCP-IP stack implemented in the Linux kernel. They are thus unsuitable to support user level implemented stacks like those provided by libioth.
- This library provides support for:
  - Client programs that need to query DNS servers
  - DNS servers, forwarders, filters that need to parse DNS queries, compose and send back appropriate replies
- Es:

```
struct iothdns *idd = iothdns_init_strcfg(stack, "nameserver 1.1.1.1");
int s = iothdns_getaddrinfo(idd, argv[1], argv[2], &hints, &result);
…
iothdns_fini(idd);
```

IoTh ecosystem 6: iothnamed

- iothnamed is a DNS *server/forwarder/cache* for the Internet of Threads supporting hash based IPv6 addresses and OTIP, i.e. one time IP.
- Main subject of this presentation.

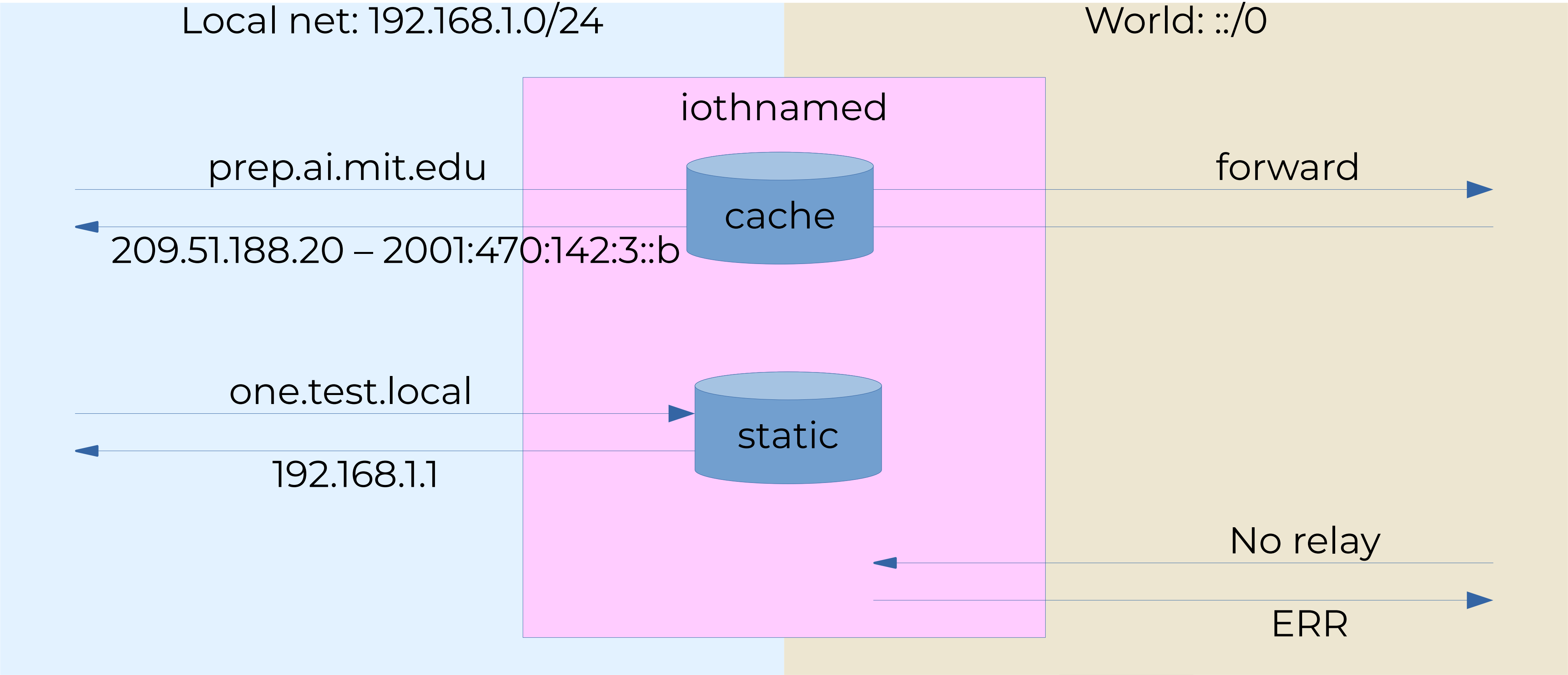
Iothnamed configuration in a nutshell

- `rstack, fstack, stack`: define stack(s) for requests, forwarding, both.
- `dns`: address of server(s) for forwarding
- `net`: define networks (address spaces)
- `auth`: define services: which domains are involved and which net(s) can access them
  - `accept`: permit TCP connections
  - `error`: return an error
  - `static`: provide static data
  - `hash`: provide a hash computed IPv6 addr
  - `hrev`: permit hash reverse resolution
  - `otip`: provide one-time IPv6 addresses
  - `cache`: grant access to the cache
  - `fwd`: allow request forwarding
- `static`: define static records (A, AAAA, PTR, CNAME, NS, MX, TXT)
- `options`: other configuration parameters

iothnamed: 1st scenario local+proxy

- static local names + proxy + cache
- iothnamed dns server runs as a caching proxy for local clients. The server also defines some local names for direct and reverse resolution.

iothnamed: local+proxy



iothnamed local+proxy config (1)

```
# The service is provided for queriers reaching this server on the
# vde network vde:///tmp/hub, IP address 192.168.1.24.
rstack    stack=vdestack,vnl=vde:///tmp/hub
rstack    mac=80:01:01:01:01:01,eth
rstack    ip=192.168.1.24/24
# The kernel stack is used to forward requests to remote dns servers
fstack    stack=kernel

# forward requests using IPv4 packets to 8.8.8.8 or 80.80.80.80
dns       8.8.8.8
dns       80.80.80.80
```

iothnamed local+proxy config (2)

```
# the net name 'local' defines the ip range 192.168.1.0/24
net       local 192.168.1.0/24

# clients from 'local' are allowed to send tcp dns requests
auth      accept local
# clients from 'local' can receive replies for names xxxx.test.local
auth      static local .test.local
# clients from 'local' can receive replies for names 1.168.192.in-addr.arpa
auth      static local 192.168.1.0/24
# search in the cache (forwarded query results are cached)
auth      cache local .
# requests from 'local' can be forwarded
auth      fwd local .
```

iothnamed local+proxy config (3)

```
# static definitions
static     A one.test.local 192.168.1.1
static     A two.test.local 192.168.1.2
# static definitions for reverse resolution
static     PTR 192.168.1.1 one.test.local
static     PTR 192.168.1.2 two.test.local
```

iothnamed local+proxy: test

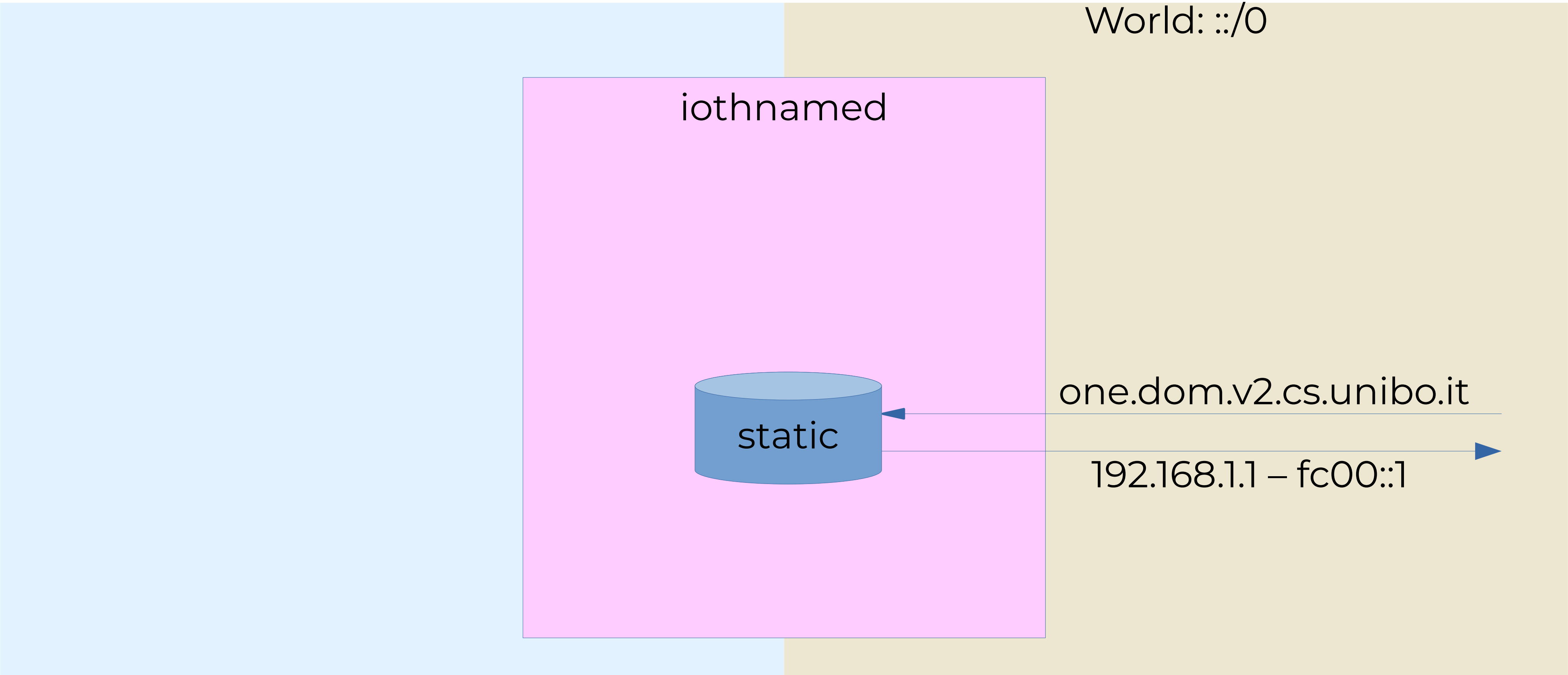
```
# from 192.168.1.1
$ host prep.ai.mit.edu
prep.ai.mit.edu is an alias for ftp.gnu.org.
ftp.gnu.org has address 209.51.188.20
ftp.gnu.org has IPv6 address 2001:470:142:3::b
$ ping one.test.local
PING one.test.local (192.168.1.1) 56(84) bytes of data.
bytes from one.test.local (192.168.1.1): icmp_seq=1 ttl=64 time=0.038 ms
bytes from one.test.local (192.168.1.1): icmp_seq=2 ttl=64 time=0.061 ms
```

iothnamed 2nd scenario

Delegated subdomain

- In this example the domain dom.v2.cs.unibo.it has been delegated to the public IP addresses 130.136.31.250 and 2001:760:2e00:ff00::fd
- (in order to test this example on your environment, IP addresses and domain names should be modified to be consistent with your scenario)

iothnamed: delegated-subdomain



iothnamed delegated-subdomain config

```
rstack    stack=vdestack,vnl=vde:///tmp/hub
rstack    mac=80:01:01:01:01:01,eth
rstack    ip=130.136.31.250/24,gw=130.136.31.1
rstack    ip=2001:760:2e00:ff00::fd/64,ip=2001:760:2e00:ff00::ff/64

# the name 'world' matches any IPv6 or IPv4 address.
net       world ::/0

# the static definition for names xxxx.dom.v2.cs.unibo.it are available for everybody
auth      static world .dom.v2.cs.unibo.it

static    A one.dom.v2.cs.unibo.it 192.168.1.1
static    AAAA one.dom.v2.cs.unibo.it fc00::1
static    A two.dom.v2.cs.unibo.it 192.168.1.2
static    AAAA two.dom.v2.cs.unibo.it fc00::2
```

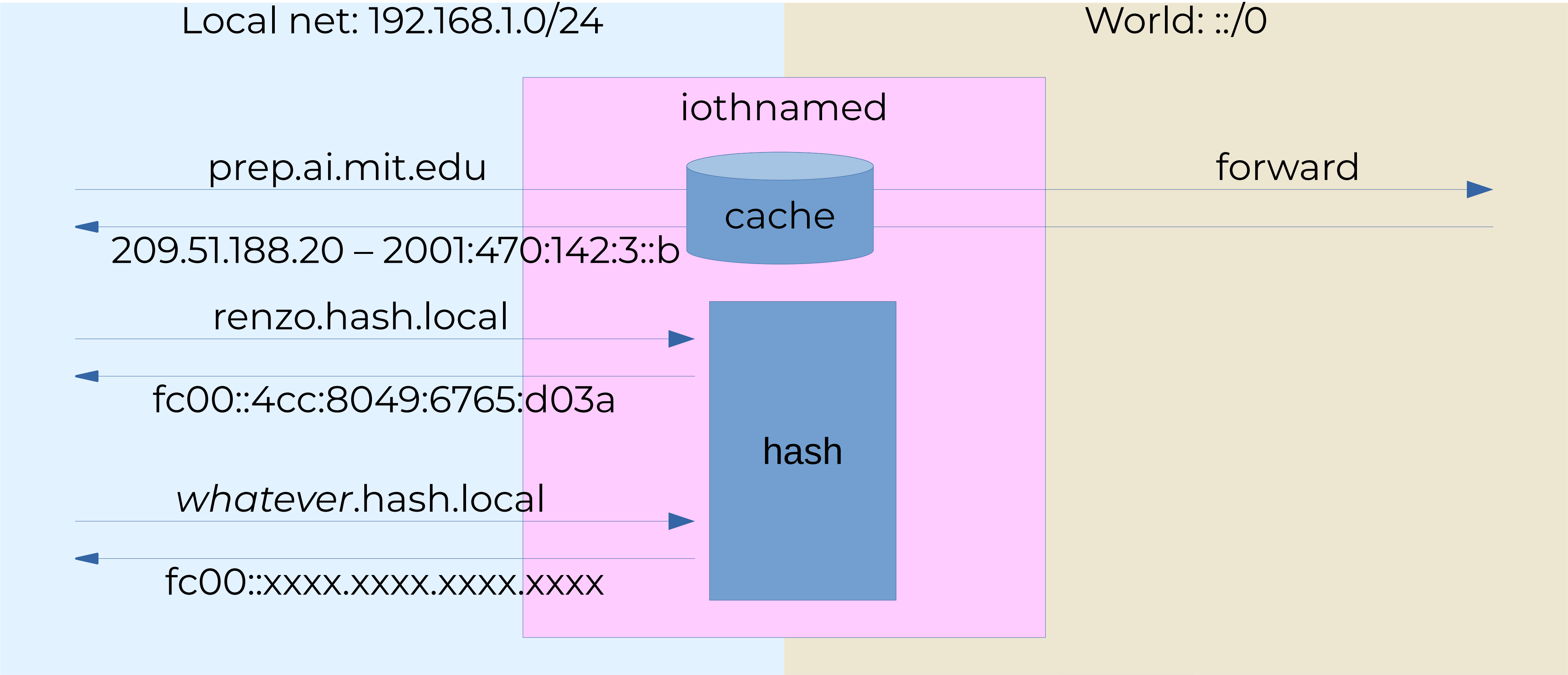
iothnamed delegated-subdomain: test

```
# From a random host connected to the Internet:


$ host one.dom.v2.cs.unibo.it
one.dom.v2.cs.unibo.it has address 192.168.1.1
one.dom.v2.cs.unibo.it has IPv6 address fc00::1
```

iothnamed 3rd scenario localhash+proxycache

- Local hash based addresses
- + proxy/cache

iothnamed: localhash+proxycache



iothnamed localhash+proxycache config (1)

```
rstack    stack=vdestack,vnl=vde:///tmp/hub
rstack    mac=80:01:01:01:01:01,eth
rstack    ip=192.168.1.24/24
rstack    ip=fc00::24/64
fstack    stack=kernel

dns       8.8.8.8
dns       80.80.80.80

net       local 192.168.1.0/24
net       local fc00::/64
auth      accept local
```

iothnamed localhash+proxycache config (2)

```
# define the base address as a static record
auth      static local hash.local
static    AAAA hash.local fc00::
auth      hash local .hash.local hash.local
auth      hrev local hash.local/64

# alt. without static definition of the base addr:
# auth      hash local .hash.local fc00::
# auth      hrev local fc00::/64

auth      cache local .
auth      fwd local .

option hrevmode always
```

iothnamed localhash+proxycache: test

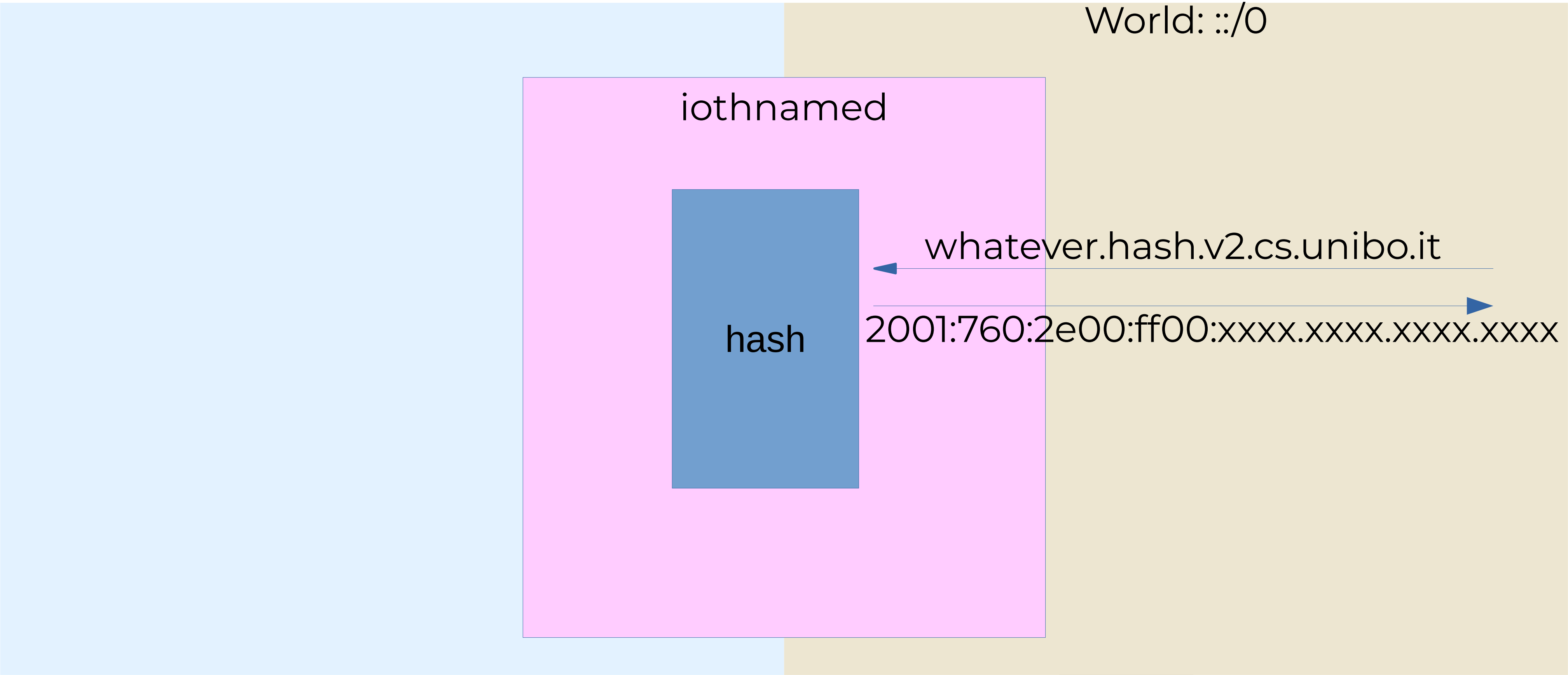
```
# from fc::1

$ host renzo.hash.local

renzo.hash.local has IPv6 address fc00::4cc:8049:6765:d03a

$ host hic_sunt_leones.hash.local

hic_sunt_leones.hash.local has IPv6 address
fc00::9c8f:74b4:705f:6512

$ host fc00::9c8f:74b4:705f:6512

2.1.5.6.f.5.0.7.4.b.4.7.f.8.c.9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
c.f.ip6.arpa domain name pointer hic_sunt_leones.hash.local.
```

iothnamed 4th scenario delegated+hash

hash based IPv6 addresses (with delegation)

- The scenario is the combination of the two previous examples. In this case the domain hash.v2.cs.unibo.it has been delegated to 2001:760:2e00:ff00::fd and 130.136.31.253, while the reverse resolution of 2001:760:2e00:ff00::/64 has been delegated to 2001:760:2e00:ff00::ff.

iothnamed: delegated+hash


World: ::/0
iothnamed
hash
whatever.hash.v2.cs.unibo.it
2001:760:2e00:ff00:xxxx.xxxx.xxxx.xxxx

iothnamed delegated+hash config

```
rstack    stack=vdestack,vnl=vde:///tmp/hub
rstack    mac=80:01:01:01:01:02,eth
rstack    ip=130.136.31.253/24,gw=130.136.31.1
rstack    ip=2001:760:2e00:ff00::fd/64,ip=2001:760:2e00:ff00::ff/64
net       world ::/0

# define glue record (for base address)
auth      static world hash.v2.cs.unibo.it
static    AAAA hash.v2.cs.unibo.it 2001:760:2e00:ff00::

auth      hash world .hash.v2.cs.unibo.it hash.v2.cs.unibo.it
auth      hrev world hash.v2.cs.unibo.it/64

option hrevmode always
```

iothnamed delegated+hash: test

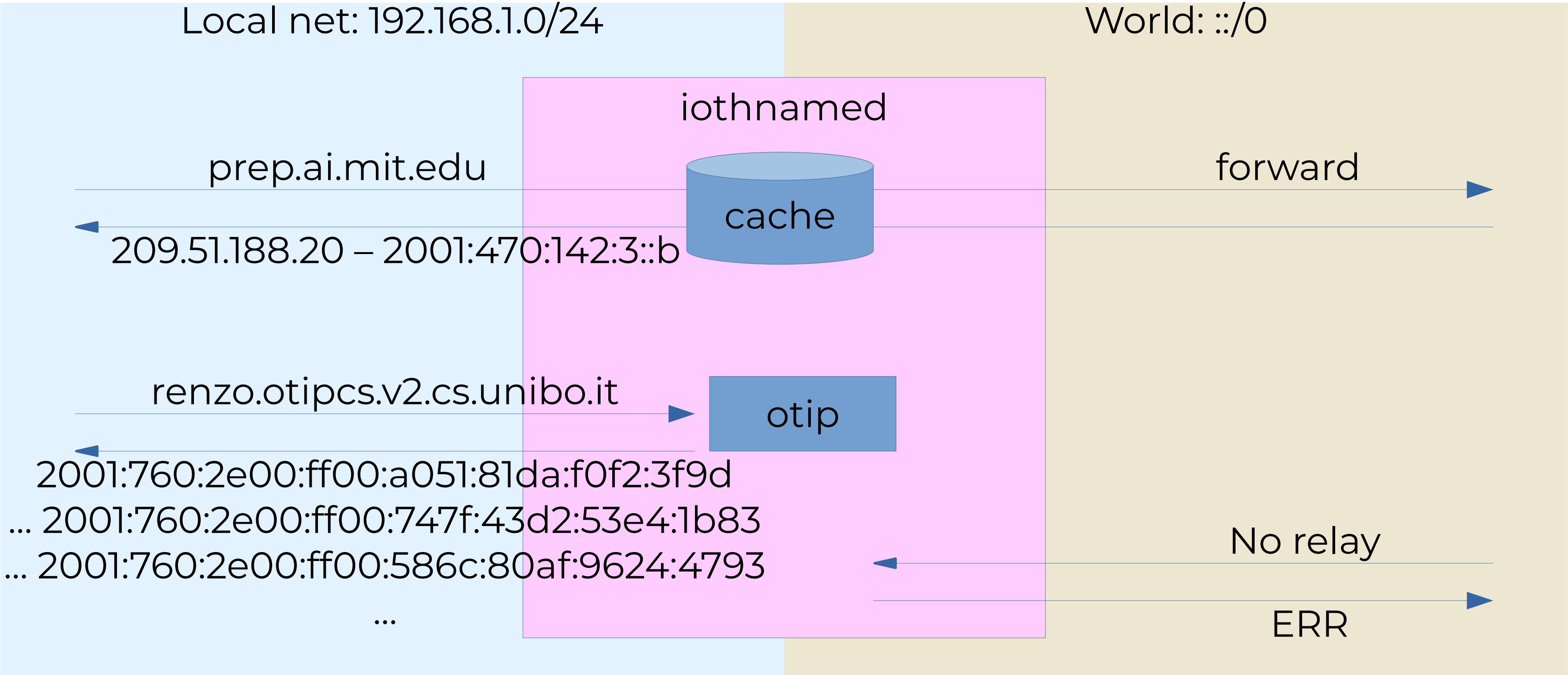
```
$ host renzo.hash.v2.cs.unibo.it

renzo.hash.v2.cs.unibo.it has IPv6 address
2001:760:2e00:ff00:6066:4f84:db3e:c9cb

$ host lucia.hash.v2.cs.unibo.it

lucia.hash.v2.cs.unibo.it has IPv6 address
2001:760:2e00:ff00:cf1:1fe9:aad4:e838

$ host whatever-you-want.hash.v2.cs.unibo.it

whatever-you-want.hash.v2.cs.unibo.it has IPv6 address
2001:760:2e00:ff00:542d:ffcb:17e:8fa7

$ host 2001:760:2e00:ff00:542d:ffcb:17e:8fa7

7.a.f.8.e.7.1.0.b.c.f.f.d.2.4.5.0.0.f.f.0.0.e.2.0.6.7.0.1.0.0.2.ip6.arpa
domain name pointer whatever-you-want.hash.v2.cs.unibo.it.
```

iothnamed 5th scenario OTIP+proxycache

OTIP: One Time IP

- iothnamed dns server runs as a caching proxy for local clients + provides the current OTIP addresses (only for the local environment).

iothnamed: fifth scenario OTIP+proxycache


Local net: 192.168.1.0/24
World: ::/0
iothnamed
prep.ai.mit.edu
cache
forward
209.51.188.20 – 2001:470:142:3::b
renzo.otipcs.v2.cs.unibo.it
otip
2001:760:2e00:ff00:a051:81da:f0f2:3f9d
... 2001:760:2e00:ff00:747f:43d2:53e4:1b83
... 2001:760:2e00:ff00:586c:80af:9624:4793
...
No relay
ERR

iothnamed OTIP+proxycache config

```
rstack    stack=vdestack,vnl=vde:///tmp/hub
rstack    mac=80:01:01:01:01:01,eth
rstack    ip=192.168.1.24/24
rstack    ip=fc00::24/64
fstack    stack=kernel
dns       8.8.8.8
Dns       80.80.80.80

net       local 192.168.1.0/24
net       local fc00::/64
auth      accept local

auth      otip local .otipcs.v2.cs.unibo.it otipcs.v2.cs.unibo.it mypassword
# auth      otip local .otip 2001:760:2e00:ff00:: mypassword

auth      cache local .
auth      fwd local .
```

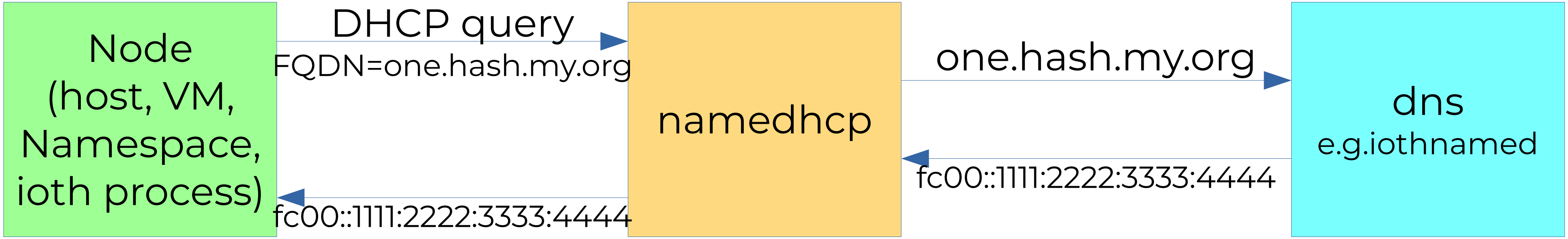
iothnamed OTIP+proxycache: test

```
# host renzo.otipcs.v2.cs.unibo.it
renzo.otipcs.v2.cs.unibo.it has IPv6 address 2001:760:2e00:ff00:6ca6:616:5145:547a

… wait ~32 secs
# host renzo.otipcs.v2.cs.unibo.it
renzo.otipcs.v2.cs.unibo.it has IPv6 address 2001:760:2e00:ff00:a051:81da:f0f2:3f9d

… wait ~32 secs
# host renzo.otipcs.v2.cs.unibo.it
renzo.otipcs.v2.cs.unibo.it has IPv6 address 2001:760:2e00:ff00:747f:43d2:53e4:1b83
```

IoTh ecosystem 7: namedhcp

- namedhcp is an IPv6 DHCP server implementation for IPV6 stateful autoconfiguration. When namedhcp receives a DHCP query including the fqdn option (option 39 as defined in RFC4704) it queries the DNS for an AAAA record. If there is such a record, the IPv6 address is returned to the DHCP client.

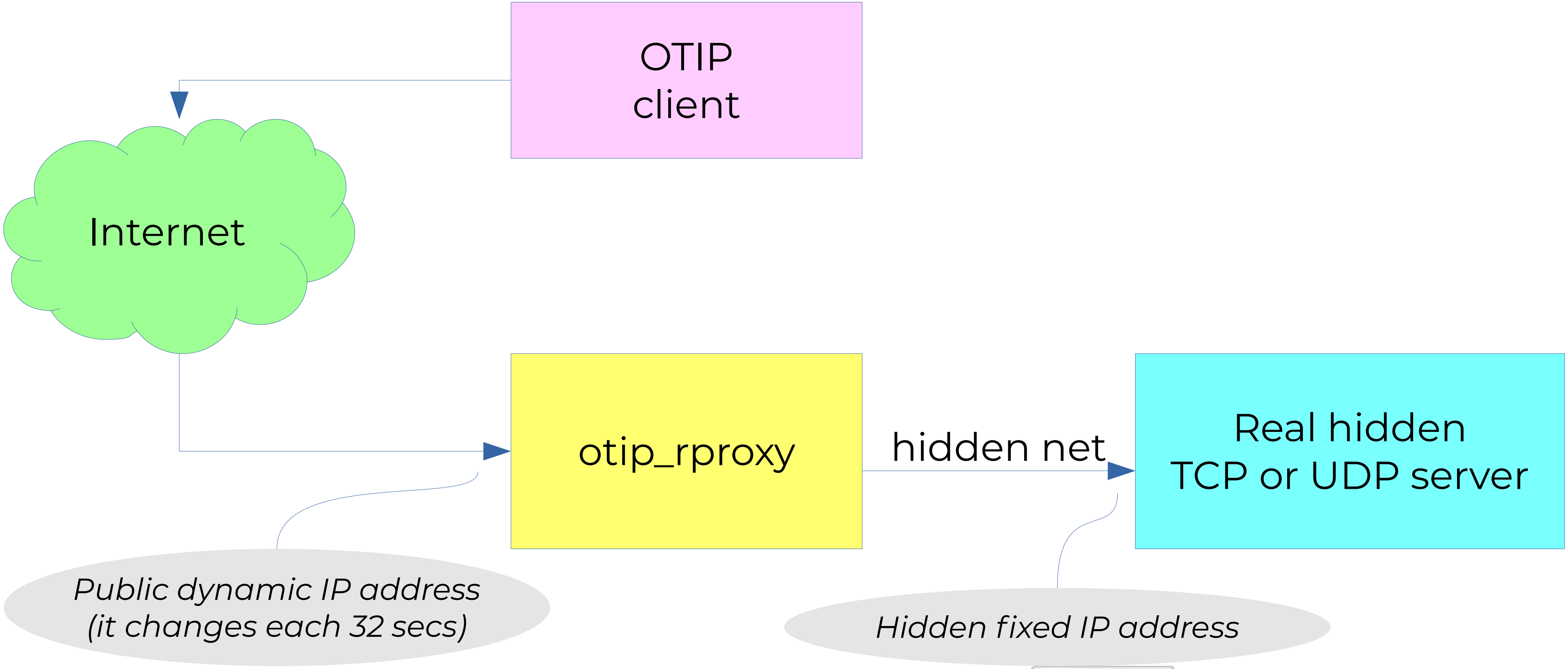
Namedhcp


Node
(host, VM,
Namespace,
ioth process)
DHCP query
FQDN=one.hash.my.org
fc00::1111:2222:3333:4444
namedhcp
one.hash.my.org
fc00::1111:2222:3333:4444
dns
e.g.iothnamed

IoTh ecosystem 8: otip-utils

- One Time IP address (OTIP) utilities
- OTIP means that the current IP address of a server changes periodically to prevent networking attacks. This method has been designed for IPv6 networks. The current IP address of a server is computed on the basis of its fully qualified domain name, some private information shared by legitimate users and the server itself, like a password, and the current time.
- otip-utils implements the following commands:
  - otipaddr: computes the current OTIP address.
  - hashaddr: computes the hash based address.
  - otip_rproxy: a OTIP enabled reverse proxy. This tool permit to protect TCP or UTP servers using OTIP

otip_rproxy



OTIP client

Internet

otip_rproxy

hidden net

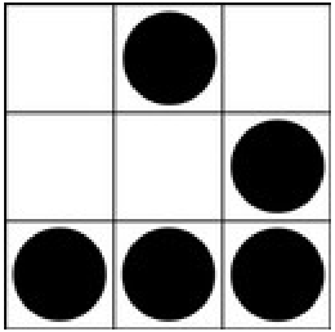
Real hidden TCP or UDP server

Public dynamic IP address (it changes each 32 secs)

Hidden fixed IP address

We are still creating art and beauty on a computer:

the art and beauty of revolutionary ideas translated into (libre) code...



renzo, rd235, iz4dje